Ваш первый CSS Grid Layout

Двумя основными компонентами CSS Grid являются обертка (parent) и элементы (children). Обертка – это сетка, внутри которой как раз и содержатся все элементы.

Вот пример разметки для parent с шестью children:

<div class="wrapper">

 <div>1</div>

 <div>2</div>

 <div>3</div>

 <div>4</div>

 <div>5</div>

 <div>6</div>

</div>

Теперь предоставляем обертке **div** отображение сетки:

.wrapper {

 display: grid;

}

Да, сейчас это не работает, но только потому, что еще не определили вид нашей сетки. Пока 6 **div** просто располагаются друг над другом:



Здесь также добавлены стили, но пусть они вас не путают: стили и [CSS эффекты](https://proglib.io/p/8-css-tricks/) никак не соприкасаются с нашей темой.

Разделяем на столбцы и строки

Давайте определим столбцы и строки, чтобы создать двухмерность. Пусть это будет 2 строки и 3 столбца. Для этого обратимся к **grid-template-column** и **grid-template-row**:

.wrapper {

 display: grid;

 grid-template-columns: 100px 100px 100px;

 grid-template-rows: 50px 50px;

}

Мы указали, что у **grid-template-columns** три значения, следовательно получим столько же столбцов. Аналогичная ситуация со строками: 2 значения в  **grid-template-rows** – это две строки. Наши строки будут задавать ячейкам ширину в 50px, а столбцы – длину в 100px. Имеем следующую разметку:



Давайте проверим, правильно ли вы поняли связь значений и вида нашей сетки. Обратимся к следующему примеру:

.wrapper {

 display: grid;

 grid-template-columns: 200px 50px 100px;

 grid-template-rows: 100px 30px;

}

Постарайтесь нарисовать это в уме. Выглядит вторая разметка так:



Располагаем элементы

А вот размещение элементов в сетке – это уже задача посложнее. Нужно очень хорошо представлять, как это все будет выглядеть. Например, создадим сетку 3х3, пользуясь той же разметкой, что и раньше:

.wrapper {

 display: grid;

 grid-template-columns: 100px 100px 100px;

 grid-template-rows: 100px 100px 100px;

}

Это создаст следующий макет:

Беда в том, что мы видим вариант сетки 3х2, тогда как определяли ее как 3х3. Все потому, что у нас изначально только 6 элементов для заполнения, поэтому еще одну строку мы попросту не видим. Чтобы она стала видимой, нам нужно добавить еще 3 дополнительных элемента.

Чтобы изменить размер или позиционировать элементы, мы воспользуемся их свойствами **grid-row** и **grid-column**:

.item1 {

 grid-column-start: 1;

 grid-column-end: 4;

}

Мы выставляем условие, при котором **item1** должен начинаться уже в первой линии и заканчиваться в 4-ом столбце. Таким образом, **item1** займет всю первую строку, и мы получим следующую разметку:

Наверное, вас терзает вопрос, почему мы указали в коде 4 столбца, когда у нас их 3? Ну тут все просто: мы говорим не о середине этих столбцов, а о закрывающих их линиях. Посмотрите на следующее изображение, чтобы понять:



Теперь мы работаем со всеми строками в сетке, которые нам видны. Грубо говоря, первый элемент, занявший всю строку, сдвинул последующие вниз, чем и заполнил все данные в условии строки.

Но на этом интересные возможности CSS Grid не заканчиваются. Вот более лаконичный и удобоваримый синтаксис примера выше:

.item1 {

 grid-column: 1 / 4;

}

Давайте посмотрим, насколько правильно вы поняли суть наших манипуляций с разметкой. Перестраиваем содержимое:

.item1 {

 grid-column-start: 1;

 grid-column-end: 3;

}

.item3 {

 grid-row-start: 2;

 grid-row-end: 4;

}

.item4 {

 grid-column-start: 2;

 grid-column-end: 4;

}

И вот что получаем в итоге. Подумайте, каким образом мы сделали сетку такой:



Базовая адаптивность с единицей измерения fr (fractional unit)

CSS Grid приносит с собой совершенно новое значение, называемое долей единицы. Доля единицы записывается как fr, и она позволяет разбить контейнер на столько частей, сколько захотите.

наша начальная сетка:



Вот HTML:

<div class="container">

 <div>1</div>

 <div>2</div>

 <div>3</div>

 <div>4</div>

 <div>5</div>

 <div>6</div>

</div>

Вот CSS:

.container {

 display: grid;

 grid-template-columns: 100px 100px 100px;

 grid-template-rows: 50px 50px;

}

Давайте изменим размер каждого столбца на 1fr.

.container {

 display: grid;

 grid-template-columns: 1fr 1fr 1fr;

 grid-template-rows: 50px 50px;

}

Здесь происходит то, что сетка распределяет всю ширину на три части, и каждый из столбцов занимает по одной единице.

Если мы изменим значение grid-template-columns на 1fr 2fr 1fr, второй столбец будет в два раза шире, чем два других столбца. Общая ширина теперь равна четырем единицам fr, второй столбец занимает две из них, а остальные — по одной.

Другими словами, значение единицы fr позволяет вам легко изменять ширину столбцов.

Расширенная адаптивность

Однако приведенный выше пример не дает нам адаптивности, которую мы хотим, поскольку эта сетка всегда будет иметь ширину в три столбца. Мы хотим, чтобы наша сетка меняла количество столбцов в зависимости от ширины контейнера. Для этого вам нужно будет изучить три новые концепции.

**repeat()**
Мы начнем с функции repeat (). Это мощный способ указания столбцов и строк. Давайте возьмем исходную сетку и изменим ее используя repeat ():

.container {

 display: grid;

 grid-template-columns: repeat(3, 100px);

 grid-template-rows: repeat(2, 50px);

}

Другими словами, repeat(3, 100px) идентично 100px 100px 100px. В первом параметре указано, сколько столбцов или строк вы хотите, а второе определяет их ширину, поэтому это даст нам тот же самый макет, с которого мы начали:



**auto-fit**
Давайте вместо фиксированного количество столбцов, определим их автоматически, заменив 3 на auto-fit.

.container {

 display: grid;

 grid-gap: 5px;

 grid-template-columns: repeat(auto-fit, 100px);

 grid-template-rows: repeat(2, 100px);

}

Теперь сетка меняет количество столбцов в зависимости от ширины контейнера.

Браузер пытается разместить как можно больше столбцов шириной 100 пикселей в контейнере.

Однако, если мы сделаем все столбцы ровно на 100 пикселей, мы никогда не получим необходимую гибкость, так как они редко будут заполнять всю ширину. Как вы можете видеть на гифке выше, сетка часто оставляет пустое пространство с правой стороны.

**minmax()**
Конечный ингредиент, который нам нужен, чтобы исправить это, называется minmax (). Мы просто заменим 100px на minmax (100px, 1fr).
Вот окончательный CSS.

.container {

 display: grid;

 grid-gap: 5px;

 grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));

 grid-template-rows: repeat(2, 100px);

}

Обратите внимание, что вся адаптивность прописывается в одной строке CSS.

И, как вы видите, это прекрасно работает. Функция minmax () определяет диапазон размеров, который больше или равен **min** и меньше или равен max.

Таким образом, столбцы теперь будут всегда не менее 100 пикселей. Однако, если имеется больше свободного места, грид будет распределять место поровну каждому столбцу, поскольку столбцы будут занимать 1fr вместо 100 px.

Добавление изображений

Теперь последний шаг — добавить изображения. Это не имеет ничего общего с CSS Grid, но давайте рассмотрим код.

Мы начнем с добавления тега изображения внутри каждого элемента грида.



Чтобы изображение поместилось в элемент, мы установим его ширину и высоту, как и у самого элемента, а затем применим object-fit: cover;. Это заставит изображение покрыть весь контейнер, и браузер обрежет его, если это необходимо.

.container > div > img {

 width: 100%;

 height: 100%;

 object-fit: cover;

}

Пройдите квест <http://cssgridgarden.com/#ru>

Раскладка структуры всей страницы (главный грид)

В нашем примере для раскладки структуры напрашивается простой двух колоночный грид для средних экранов и трехколоночный грид для широких, чтоб выделить место справа под сайдбар.



«Поиграв» с шириной окна, вы найдете идеальный брейкпойнт для грида при средней ширине. Внутри этого медиавыражения укажите, что элемент .site будет грид-контейнером:

@supports (grid-area: auto) {
        @media screen and (min-width: 56.25em) {
                .site {
                        display: grid;
                        grid-template-columns: 15em 1fr;
                        grid-template-rows: 1fr minmax(1em, auto);
                }
        }
}

Чтобы легче понимать, что в гриде происходит, дадим имена разным частям грида с помощью grid-template-areas. Свойство grid-template-area дает возможность построить карту своего рода именованных прямоугольных областей в гриде, охватывающих одну и более ячеек. Как только такая область определена, грид-элемент можно привязать к ней с помощью свойства grid-area и имени области. В примере ниже свойство grid-template-area служит для создания двухколоночной раскладки, определяя всю первую колонку как область header и деля вторую колонку на области main, sidebar и footer:

@supports (grid-area: auto) {
        @media screen and (min-width: $query\_\_medium) {
                .site {
                        display: grid;
                        grid-template-columns: 15em 1fr;
                        grid-template-rows: 1fr minmax(1em, auto);
                        grid-template-areas: "header main"
                                                   "header sidebar"
                                                   "header footer";
                }
        }
}

Теперь можно разместить шапку, основной контент, сайдбар и подвал в именованных областях шаблона. И каждому, читающему этот CSS, будет понятно, что тут происходит:

@supports (grid-area: auto) {
        @media screen and (min-width: $query\_\_medium) {
                .site {
                        display: grid;
                        grid-template-columns: 15em 1fr;
                        grid-template-rows: 1fr minmax(1em, auto);
                        grid-template-areas: "header main"
                                                   "header sidebar"
                                                   "header footer";
                }
                .site-header {
                        grid-area: header;
                }
                .site-main {
                        grid-area: main;
                }
                .widget-area {
                        grid-area: sidebar;
                }
                .site-footer {
                        grid-area: footer;
                }
        }
}

Для самых широких экранов будет своё медиавыражение. Здесь сайдбар занимает новую колонку по правой стороне грида. Для этого нужно изменить шаблон грида, чтобы поместить сайдбар в новую колонку. Поскольку сайдбар уже привязан к области шаблона «sidebar», нам нужно лишь изменить правило для .site:

@supports (grid-area: auto) {
        @media screen and (min-width: 56em) {
                .site {
                        display: grid;
                        grid-template-columns: 15em 1fr;
                        grid-template-rows: 1fr minmax(1em, auto);
                        grid-template-areas:
                                "header main"
                                "header sidebar"
                                "header footer";

                        @media screen and (min-width: 70em) {
                                grid-template-columns: 15em 1fr 15em;
                                grid-template-rows: 1fr minmax(1em, auto);
                                grid-template-areas:
                                        "header main sidebar"
                                        "header footer footer";
                        }
                }
                .site-header {
                        grid-area: header;
                }
                .site-main {
                        grid-area: main;
                }
                .widget-area {
                        grid-area: sidebar;
                }
                .site-footer {
                        grid-area: footer;
                }
        }
}

Используя @supports (grid-area: auto) { }, вы можете где угодно в файле создавать добавочные CSS-правила, которые сработают только если браузер поддерживает грид-раскладку.